AMATH 482 Homework 3

Gonzalo Ferrandez Quinto

February 22, 2020

1 Introduction and Overview

On this writing I will be analyzing the the movement of a spring-mass system with a series of complications in the data that I will expose later. The premise is that we are given camera recordings of the spring in different angles and using that data, I need to recreate the motion of the mass. For this task I will be using the singular value decomposition (SVD) to perform a principal component analysis (PCA). I will explain what this entails in the theoretical background section. The data will be analyzed with the input of three different cameras in four conditions:

1. **Ideal Case:** Here the mass is only moving in the z-direction in a simple harmonic oscillation. In addition, all the cameras are held in place.
2. **Noisy Case:** The situation will be the same as before, with the exception that now the cameras are being held by hand and are are shaken a little. Therefore incorporating movement noise into the recordings.
3. **Horizontal Displacement:** The mass is now released off-center and produces both movement in the x-y plane and the z-direction.
4. **Horizontal Displacement and rotation:** In this case the mass is released off-center and it also rotates in the x-y plane while it also moves in the z-direction.

Therefore in this writing I will be exploring these four different scenarios and comparing the results they produce. The data will be formed by pixels that vary over time, therefore each one of the cameras will give information about two directions of the oscillation. In addition, the mass has attached to it a flashlight which will make it easier to locate its position using the pixel intensities.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

The SVD is a tool that decomposes any matrix in a series of matrices multiplication that just involve rotations and stretches. In addition, the SVD is valid also for non-square matrices. Therefore any matrix of the form $m \times n$ can be decomposed by:

$$A = U\Sigma V^* \tag{1}$$

Here U is a $m \times m$ matrix and V^* is a $n \times n$ matrix, where both represent rotations. As these are rotation matrices, it will mean that they are unitary and therefore for both of them $V^{-1} = V^*$ (inverse equals the transpose). Also Σ is a real postive diagonal matrix of the form $m \times n$ which represents the stretch. In addition, its diagonal values are ordered in descending order from the highest to lowest. In the SVD basis then, the columns of the matrix U represents the principal directions of stretch that are occurring when the matrix A is applied as a linear transformation. In addition, the diagonal values of the matrix Σ represent how much a vector is being stretched in this direction.

2.1.1 SVD computation

It can be shown that for some matrix A, the the matrix V from the SVD can be calculated by solving the eigenvalue equation:

$$A^*A = V\Sigma^2 \tag{2}$$

In addition for matrix U, it can be calculated by solving the eigenvalue equation:

$$AA^* = U\Sigma^2 \tag{3}$$

In both cases the eigenvalues will be represented by the square of the diagonal matrix in the SVD.

2.2 Norm and Low-dimensions approximation

The two norm of matrix is defined as:

$$\|A\|_2 := sup\|Ax\|_2 \tag{4}$$

Where the vector x is a unit vector. Therefore the 2 norm of a matrix will be the maximum possible stretch a unit vector can have when transforming it with matrix A. If we put matrix A in the form $A = U\Sigma V^*$ we can see that both U and V^* will not contribute to the norm as they are rotation matrices only. Therefore the biggest possible stretch will be the first diagonal term in matrix Σ (σ_1) which is known as the first singular value. Then $\|A\|_2 = \sigma_1$.

In addition if A of size $n \times m$ is a matrix of rank r, then it can be represented by the sum:

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^* \tag{5}$$

Here all the matrix terms j of U and V^* correspond to columns of matrices, while σ_j will be the jth diagonal entry in Σ. Being the multiplication of $u_j v_j^*$ (outer product) with also size $n \times m$. In addition an outer product will always have rank one. Therefore A can be represented as sum of rank 1 matrices of its same size.

This summation does not need to be take till rank r, one can also cut at any desired rank N. This will produce the best lower rank approximation of A of rank N. Therefore approximating A in a lower-dimensional matrix. It can also be shown that out all of the possible matrices of rank N, the minimum 2 norm error will be given by $\|A_r - B_N\|_2 = \sigma_{N+1}$. Therefore being $B_N = A_N$ and A_N is the N rank approximation of A using the SVD entries.

2.3 Principal Component Analyisis

If we put all the camera recording measurements organized by camera entry andd time we get:

$$X = [x_1 y_1]' \tag{6}$$

Where each of the entries are column vectors that have N mesurements in time and correspond to the x-y pixel coordinates of the camera.

From X, a covariance matrix can be created where $C_X = XX^T = \begin{bmatrix} \sigma^2_{x_1} & \sigma^2_{x_1 y_1} \\ \sigma^2_{y_1 x_1} & \sigma^2_{y_1} \end{bmatrix}$ Here the diagonal entries represent the spread in each one of the dimensions of matrix X, (the bigger the more spread) and the off-diagonal entries represent how much does the data correlate in those two dimensions (zero do not correlate, 1 they are the same).

In a system like the spring, we would like that this off-diagonals where zero. Therefore we would not have any correlation in the different directions in the data and we could express it in an orthogonal basis. With each one of the individual variances giving the importance of that direction.

Making all of the off-diagonal zero could be accomplished by doing a change of basis. It can be shown that if we define our change of basis as $Y = U^T X$, the co-variance is now $C_Y = \Sigma^2$. Where U is the matrix U transposed from the SVD of X and Σ^2 are the singular values squared. To give the values, Σ needs to be scaled by a factor of $\frac{1}{\sqrt{n-1}}$ Where n is the length of each row vector in X. Now if we just keep the first low-rank approximation, we can check with the variance the importance of each one of these and the maximum norm 2 error we will have. Therefore just keeping the first row in Y, will keep the principal movement of the spring that has the highest variance.

3 Algorithm Implementation and Development

3.0.1 Getting the coordinates positions

In order to produce the SVD for the PCA, I first needed to create the 2-dimensional position data matrix for each one of the cameras. To do these I used 2 methods:

For both cases, in every step of this analysis I used the command `rgb2gray()` to put each one of the camera frame data in a gray scale. In addition, there was a flashlight on top of the mass that could be used to help with the location of the mass. In grayscale you also know that higher values in the matrix data correspond to brighter points.

For the ideal case, I knew that the relative pixel position of the flashlight compared to the previous location would be proportional to the pixel movement. For the first location of the mass I used the command `ginput()` which lets you introduce graphically coordinate positions based on the first frame. After that I iterated over each one of the frames and created a 41x41 array with the previous maximum value at the center. After that I used the command `max()` to find the point in this matrix with the highest value. After this I used the command `ind2sub()` to find the subscript of this value. I found the difference with the center of the matrix, therefore the new position value was the previous plus this difference. then repeated this for each one of the cameras.

For the rest of the cases due to the non-predictable or noisy data I took another approach. First, for all the camera frames I took part of the background and make it black. This was in the regions where I new the mass would never be, based on the videos produced by the `implay()`. After this I found all the pixels indices that had an intensity value over 240 in the gray scale using the command `find()`. Then I found the subscript averaged the indices and rounded them down to produce the location using the commands `ind2sub()` and `round()`

3.0.2 Aligning the data

In this case not in all cameras the mass started at the same position, therefore as this is a periodic movement, I trimmed the start of the data till the can located itself in the top most location in each of the cameras by using the command `min()`.Then I also trimmed the end of the data to make all of it of equal length.

3.0.3 PCA

Finally to produce the PCA analysis of the data I first computed the reduced SVD(without all the zeros) using the command `svd(,'econ')`. This gives you the U,Σ and matrices. After that I computed the energies of the using the matrix Σ by squaring each one the entries and normalizing it over the sum of the square of the trace. This allowed me to see the importance of each one the lower-rank aproximations depending on

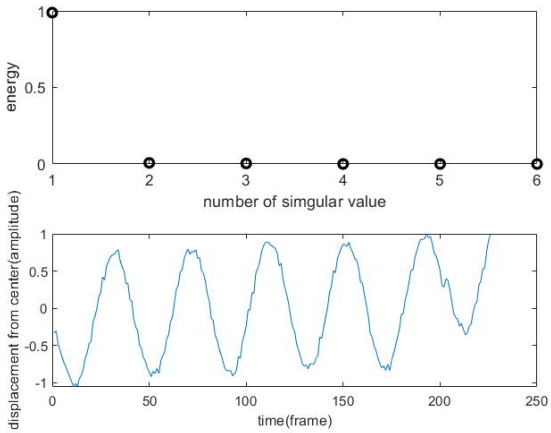

Figure 1: First rank approximation for the ideal case, with energy of 0.9888

the case. In addition, I also plotted the oscillatory movement produced by each one the cameras and the projection onto the principal movement direction by taking the first row of $Y = U^T * X$

4 Computational Results

For the ideal case I got a really good result for a rank 1 approximation with 0.9888 of all the energy in the system. This is understandable as a rank 1 approximation should be sufficient to express the data as the object is only moving in the z-direction. In addition, the rest of the energies have a massive drop-off.

For the noisy case, I still was able to do a rank1 approximation that contained the majority energy. In this case it had 0.9884 of the total. The only problem in this case is the due to the noise introduced when shaking the camera, I did not get a smooth graph like in case one although it approximates it. In addition, is still can be approximated using a rank1 summation with almost the same energy as the ideal case what is incredible taking into account all the noise.

For case three, you can see that in figure three, that changing from rank 1 to rank 2 actually does a significant improvement. While doing a rank 3 approximation does not give much more detail over rank 2. Therefore a 2-dimensional approximation is good enough in this case. WE can also see that energy is mostly contained in the first direction, although a 2 dimensional approximation improves the result. No we need to dimensions to approximate the movement.

For the last case after performing again a rank1, rank2 and rank3 approximations, it can be seen that in all cases its improving. In this case we actually get a graph improvement from rank 2 to rank 3 although still most of the energy is in the rank 1 approximation. This could mean that the movement now needs to be approximated using three dimensions. Which could be explained by the rotation.

5 Summary and Conclusions

During this whole project, the value of the SVD can be seen. It can approximate high dimensional data and actually take noise and eliminate redundancy in the process. Although this is a simple case for the application of the SVD, more uses can be found for it in almost every field in applied math.

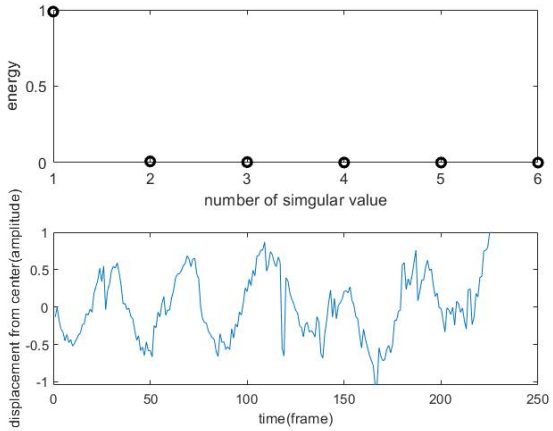

Figure 2: Rank1 approximation of the noisy case, with energy of 0.9884

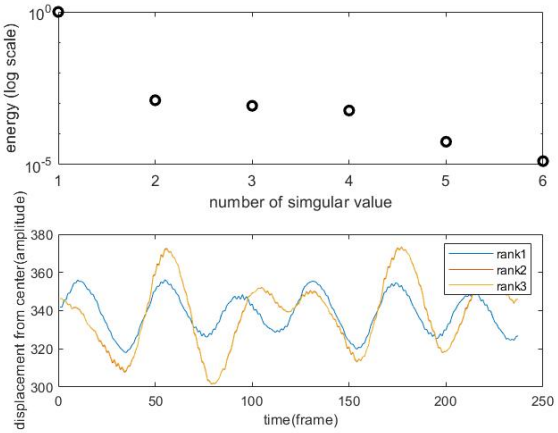

Figure 3: Improving rank approximations of case 3

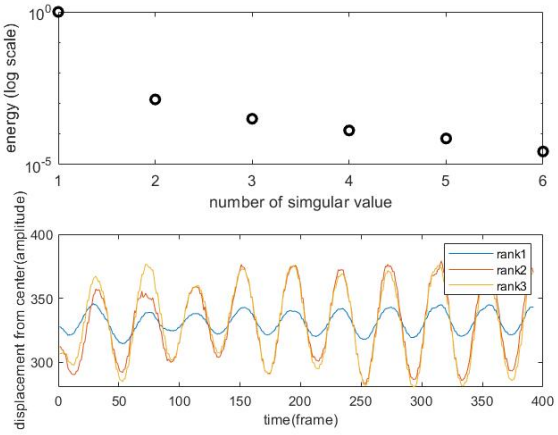

Figure 4: Improving rank approximations of case 4

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `svd()` performs the SVD calculation on matrix.
- `ind2sub()` gives you the subscripts given the index in a matrix
- `implay()`creates video from the camera input
- `max()` returns the maximum value of an array
- `find()` gives you indices of non-zero entries
- `ginput()`lets the user introduce graphical location input `fin()`

Appendix B MATLAB Code

```
clear all
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
numFrames = size(vidFrames3_1,4);
Xt3 = zeros(2,numFrames);

X = vidFrames3_1(:,:,:,1);
Xg = rgb2gray(X);
imshow(Xg);
[x3,y3] = ginput(1);
Xt3(1,1) = y3;
Xt3(2,1) = x3;

for j = 2:numFrames
    x = Xt3(1,j-1);
    y = Xt3(2,j-1);
    X = vidFrames3_1(:,:,:,j);
    Xg = rgb2gray(X);
    Xg1 = Xg;
   check = Xg(round(x-20,0):round(x+20,0),round(y-20,0):round(y+20,0));
    [C,I] = max(check(:));
    [row,col] = ind2sub(size(check),I);
    Xt3(1,j) = x+row -21;
    Xt3(2,j) = y + col-21;
end
```

Listing 1: fuction that finds positions in the ideal case

```
z = zeros(480,640);
load('cam1_2.mat');
numFrames = size(vidFrames1_2,4);
Xt1 = zeros(2,numFrames);
ps = zeros(2,1)
for q = 1:numFrames
X = vidFrames1_2(:,:,:,q);
Xg = rgb2gray(X);
z(170:430,220:480) = Xg(170:430,220:480);
Xf = double(z);
bool = Xf > 250;
I = find(bool);
[xpos, ypos] = ind2sub(size(bool),I);
v = [mean(xpos); mean(ypos)];
Xt1(:,q) = v ;
end

load('cam2_2.mat');
numFrames = size(vidFrames2_2,4);
Xt1 = zeros(2,numFrames);
ps = zeros(2,1)
for q = 1:numFrames
X = vidFrames2_2(:,:,:,q);
Xg = rgb2gray(X);
z(170:430,220:480) = Xg(170:430,220:480);
Xf = double(z);
bool = Xf > 210;
I = find(bool);
[xpos, ypos] = ind2sub(size(bool),I);
v = [mean(xpos); mean(ypos)];
Xt1(:,q) = v ;
end


load('cam3_2.mat');
numFrames = size(vidFrames3_2,4);
Xt1 = zeros(2,numFrames);
ps = zeros(2,1)
for q = 1:numFrames
X = vidFrames3_2(:,:,:,q);
Xg = rgb2gray(X);
z(170:430,220:480) = Xg(170:430,220:480);
Xf = double(z);
bool = Xf > 240;
I = find(bool);
[xpos, ypos] = ind2sub(size(bool),I);
v = [mean(xpos); mean(ypos)];
Xt1(:,q) = v ;
end
```

Listing 2: fuction that finds positions for all cases except the ideal case

```
minl = min([length(Xt1(1,:)) length(Xt2(1,:)) length(Xt3(1,:))])
t = 1:minl

Xt1 = Xt1(:,1:minl);
Xt2 = Xt2(:,1:minl);
Xt3 = Xt3(:,1:minl);

XY = [Xt1;
    Xt2;Xt3];
[U,S,V] = svd(XY,'econ');
Y = U'*(XY);
st = 1:length(diag(S));


energies = (diag(S).^2)./(sum(diag(S).^2));
figure(1)
subplot(2,1,1)
semilogy(st,energies,'ko','linewidth',2)

set(gca,'Fontsize',10,'Xtick',1:6)
xlabel("number of simgular value")
ylabel("energy (log scale)")
rank3=U(:,1:3)*S(1:3,1:3)*V(:,1:3)';
rank2=U(:,1:2)*S(1:2,1:2)*V(:,1:2)';
rank1=U(:,1)*S(1,1)*V(:,1)';
m = rank1(1,:);
n = rank2(1,:);
p = rank3(1,:);
subplot(2,1,2)
plot(t,m)
hold on
plot(t,n)
hold on
plot(t,p)
ylabel("displacement from center(amplitude)")
xlabel("time(frame)")
legend('rank1','rank2','rank3')
```

Listing 3: function that does PCA analysis